International Journal of Engineering & Technology, 7 (2.4) (2018) 46-52

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
DOI: 10.14419/ijet.v7i2.4.10040
Research paper

A Novel Approach of Data Deduplication for Distributed Storage

Shubhanshi Singhal[1], Akanksha Kaushik [2*], Pooja Sharma [3]

[1]Assistant Professor, TERii, Kurukshetra University, Kurukshetra-136119
[2] Assistant Professor, St. Andrews Institute of Technology and Management, Gurugram, M.D.U., Rohtak-124001
[3] Lecturer, Government College for Women, Karnal-132001
**Corresponding author E-mail:er.akankshakaushik@gmail.com*

Abstract

Due to drastic growth of digital data, data deduplication has become a standard component of modern backup systems. It reduces data redundancy, saves storage space, and simplifies the management of data chunks. This process is performed in three steps: chunking, fingerprinting, and indexing of fingerprints. In chunking, data files are divided into the chunks and the chunk boundary is decided by the value of the divisor. For each chunk, a unique identifying value is computed using a hash signature (i.e. MD-5, SHA-1, SHA-256), known as fingerprint. At last, these fingerprints are stored in the index to detect redundant chunks means chunks having the same fingerprint values. In chunking, the chunk size is an important factor that should be optimal for better performance of deduplication system. Genetic algorithm (GA) is gaining much popularity and can be applied to find the best value of the divisor. Secondly, indexing also enhances the performance of the system by reducing the search time. Binary search tree (BST) based indexing has the time complexity of $\theta(\log n)$ which is minimum among the searching algorithm. A new model is proposed by associating GA to find the value of the divisor. It is the first attempt when GA is applied in the field of data deduplication. The second improvement in the proposed system is that BST index tree is applied to index the fingerprints. The performance of the proposed system is evaluated on VMDK, Linux, and Quanto datasets and a good improvement is achieved in deduplication ratio.

Keywords: *Data Deduplication, Chunking, Fingerprints, Indexing, Genetic Algorithm*

1. Introduction

The amount of digital data is increasing exponentially. It is estimated that 163.2 zettabytes of digital data will be produced in 2025 [1]. In this big data era, management of this data deluge is an important and challenging task. Data deduplication is a successful data reduction approach that manages storage space by removing redundant data [2]. Thus it has increased attention in large-scale storage systems. It identifies duplicate contents by a cryptographically secured hash signature like SHA-1, MD-5 and performs data reduction by eliminating redundant data at chunk-level. Data deduplication is performed in three key steps: chunking [3], fingerprinting [4], and indexing of fingerprints [5]. Data deduplication workflow is shown in figure 1 [6]. Chunking splits the input data stream into small pieces known as chunks. There are two different approaches to divide files into chunks: fixed size chunking and variable size chunking. Chunk size is an important factor because it decides the performance of the data deduplication system [7]. Deduplication detection ratio of variable-size chunks is better than fixed-size chunks.

Various methods like BSW [8], TTTD [9], TTTD-S [10], Byte-index [11], FastCDC [12] have been proposed to decide the chunk boundary. The size of chunks is generally decided by divisor 'D'. The value of 'D' should be optimal to get better results. Genetic algorithm is an optimization technique based on the Darwinian theory of evolution [13]. Initially, it starts with a set of randomly generated chromosomes, then the processes like fitness-based selection and crossover are carried out to produce the next generation. During mating, chromosomes are chosen either randomly or according to their fitness then crossover process recombines the genetic material to produce the new chromosomes. This process is repeated either a fixed number of times or meeting the stopping criteria. By this way, a GA "evolves" a good solution to a given problem. It is also able to discover the divisor's value for each dataset. In this process, first, one-tenth of the dataset is 'split' into subparts (the number of subparts $\propto$ size of the dataset). These subparts are called as chromosomes. By applying crossover and mutation operations, the chromosomes are optimized to get optimal value of divisor 'D'.

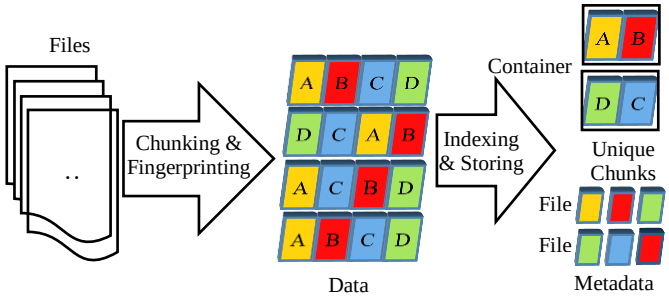

Figure 1: Data Deduplication Workflow

After chunking process, cryptographically secure hash signature (e.g. MD-5, SHA-1, SHA-256) is applied on chunks to calculate their fingerprints [14]. This process is known as fingerprinting.

Fingerprint value for each chunk is unique. Only unique chunks are placed on the disk after verifying their uniqueness using their fingerprints and redundant chunks are only referenced with old chunks. Indexing is a way to organize fingerprints on disk. Various approaches have been proposed for on-disk index-lookup process but their search time and running cost are high. Indexing should be fast enough that it can check the uniqueness of any fingerprint in minimum time. Binary search tree is a good searching algorithm having search time of $\theta(\log n)$. In this proposed work, BST indexing tree is applied to arrange and search the fingerprints. It arranges the fingerprints in the form of a tree where fingerprints can be easily checked that they are unique or not. Hadoop is an open source framework that is used to manage and perform an operation on big data [15]. It includes the HDFS, Hadoop common, Hadoop YARN, and Hadoop map-reduce. The main components of Hadoop are HDFS and map-reduce. HDFS is a master-slave architecture in which master contains information of name_node and job tracker. The slave contains the data node, task tracker, and map-reduce information. The performance analysis of proposed algorithm is performed for VMDK [16], Linux [17], and Quanto databases using Hadoop. The rest of this paper is organized as follows: In section 2, related work to data deduplication is discussed. Section 3 gives the idea of genetic algorithms. Binary search tree is discussed in section 4. Section 5 represents the algorithm for proposed architecture. The experimental results are given in section 6 and in last, the paper is concluded with a brief summary in section 7.

2. Related Work

In this world, digital data is increasing rapidly. High growth rate of data causes the storage problem because the storage space is not increasing in same proportion. Therefore, the issue arises to adjust all data in the available storage space. So, the techniques like data compression, Huffman coding, data deduplication are used to manage the data well. Data compression technique compresses the data using various compression approaches like LZO [18], LZW [19] etc. Data deduplication is faster, more scalable, and efficient than compression approaches. Data deduplication is performed in two folds. First, identification and elimination of redundancy at chunk level using techniques like Rabin [20], BUZZ [21]. Second, identification of duplicate content by calculating hash-based fingerprints using techniques like MD-5, SHA-1, SHA-256. Data deduplication removes the duplicate chunks and keeps references to old chunks.

Data deduplication was proposed in the 2000s to support global compression in large-scale storage systems at a much coarser granularity [22]. In the current scenario, data deduplication techniques are widely used to eliminate duplicacy at chunk level. First, the file-level deduplication [22] was proposed but later it was replaced by chunk-level deduplication [14] because of its better results. Low-bandwidth network file system (LBFS) was proposed to catch and remove duplicate chunks of variable size [23]. Venti was also proposed to eliminate duplicate chunks and saves storage space [14]. Variable-size chunking was employed by LBFS to find redundancy at chunk level whereas fixed size chunking was used by Venti. The chunking methods are mainly of two types: fixed size chunking and variable-size chunking. In fixed size chunking, constant size chunks are created from input data stream according to the offset of the content. A minor change in the data file can lead to the shift of boundaries for all chunk, which tends to generate a problem called as a boundary-shift problem. The chunks that contain slightly different content fail to deduplicate [23]. Variable-size chunking divides the input stream into variable-size chunks according to the content itself, this solves the problem of boundary-shift. It is most successful and widely used chunking method. Some popular variable size chunking algorithms are leap based chunking [24], bimodal chunking [25], multimodal chunking [26]. A new CDC algorithm, Asymmetric Extremum (AE) was proposed by Zhang et al. [27] that mainly focus to improve the chunking throughput and the chunk size variance. The limitations of Rabin fingerprint based CDC [20] and MAXP [28] algorithms are superseded by AE algorithm. After chunking process, the cryptographically secure hash-based signature is applied on each chunk to compute its fingerprint [29]. This process is known as fingerprinting. Fingerprinting technique simplifies the process of duplicate identification. MD-5 [30], SHA-1 [31], and SHA-256 [32] are widely used cryptographic hash-based signatures that generate fingerprints to identify the identical chunks. SHA-1 is the highly used fingerprinting algorithm because its hash collision probability is very small or to be ignored. Venti [14], LBFS [23], iDedup [33], MAD2 [34], and DDFS [35] are SHA-1 based deduplication methods. ZFS [36] and Dropbox [37] are the systems that use stronger hash algorithm i.e. SHA-256 to reduce the risk of hash collision. More than 80% of the time overhead can be attributed to chunking (about 45%) and indexing (about 35%).

3. Genetic Algorithm

John Holland proposed an idea of GA to find the solution of problems that were computationally intractable [38]. He provided the theoretical and conceptual essentials related to the design of the GA. Highly modular nature of GA makes it straightforward to implement. It is a basically an optimization technique, originally powered by the Darwin theory of evolution through the genetic solution. GA operates on a population of artificial chromosomes. GA is constructed from a number of distinct tasks. Its main tasks are the chromosomes encoding, the fitness function, selection, recombination, and the evaluation scheme. GA operates on a population of chromosomes that are string representative of solutions to an individual problem. Any particular representation used for a given problem is called chromosomes. They are basically strings of finite alphabets. Each chromosome expresses a solution to a problem and has a fitness value, which measures how good a solution is to particular problem. The working architecture of genetic algorithm is shown in figure 2 [39].

Fitness: The fitness function is a mathematical formula that checks the quality of chromosomes i.e. fitness value, as a solution for given problem.

Selection: The fitness value is a key factor in the selection of chromosomes for crossover. The selection operator in GA is applied for the selection of chromosomes for crossover on the basis of their fitness. Roulette wheel is a popular selection method that gives chance to each chromosome according to their fitness value.

Recombination/Crossover: Recombination is the process by which the selected chromosomes from parent population are recombined together using one-to-one relationship to produce new chromosomes for next generation.

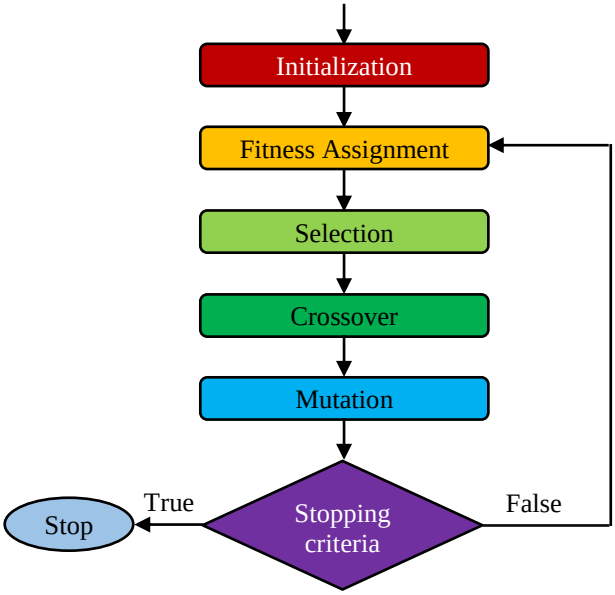

Figure 2: Working Architecture of Genetic Algorithm

Mutation: Mutation operator is applied to conserve the genetic diversity from one generation to next. The mutation value should be taken in small proportion.

Evolution: The processes of selection and recombination are repeated until the stopping criteria are not reached.

GA usually performs better than traditional techniques to calculate the value of divisor 'D'. Genetic algorithm can manage datasets with many features. They don't need specific knowledge about the problem under study.

4. Binary Search Tree

When the fingerprints or hash values are static, linear indexing is efficient. In linear indexing, fingerprints are inserted or deleted rarely. But these changes are frequent in the large-scale storage system. As the volume of data increases, the total size of fingerprints will quickly overwhelm the main memory. This arises the problem of storing and indexing of these fingerprints on the disk. Indexing also helps in determining duplicate and non-duplicate data chunks. Accessing throughput to on-disk fingerprint-index is approximately 1-6MB/s. It is a severe performance bottleneck in these systems. Therefore, a better indexing technique must be proposed that reduces search time and computational overhead. Binary search tree (BST) [40] would be a good way to store fingerprints. When BST is stored in main memory then the search and update operations are performed in $\theta(\log n)$ time. When the tree is stored on disk, the depth of tree decides the return time. It is because all the nodes along the path from the root are visited. The problem becomes greater if the BST is unbalanced. Deep nodes in the tree have the potential of causing many disk blocks to be read. So the arrangement of nodes should be in such way that search operation should be minimum. This problem can be solved by balancing the tree after few updates.

5. Proposed Work

From the above discussion, it is clear that chunking and indexing of fingerprints are very important tasks. The previously proposed chunking methods like BSW [8], TTTD [9], TTTD-S [10], Byte index chunking [11], FastCDC [12] etc. are used to calculate the size of chunks. The chunk-size is a very important factor because it directly affects the performance of deduplication system. The value of divisor 'D' decides the chunk size. How to fix the value of divisor 'D' is an important task. Small size chunks lead to high metadata overhead because overhead is proportional to the number of chunks and the problem with larger size chunks is that the deduplication ratio falls down. The previously proposed techniques do not provide the optimal value of divisor 'D' hence there is need to fulfill this gap by proposing a new method to find the optimal value of dynamic divisor 'D'. The genetic algorithm (GA) is an evolutionary technique where problems are defined in the form of chromosomes as a computer program. GA performs predefined tasks on the set of chromosomes to get best results. The same technique (GA) is applied to find the value of divisor 'D'. After the study of data deduplication literature, it is very clear that the value of divisor depends upon the redundant content i.e. Divisor $\propto$ Similarity. For example, if the similarity between two files is 25% then the chunk size should be small to get more redundancy. If the similarity between two files is 75% then large size chunks can easily find redundancy and target shifts to reduce the number of comparisons. From the above example, It is very clear that the value of divisor should be decided on the basis of similarity in datasets. Sϕrenson-Dice coefficient, a popular method, is used to calculate the similarity using the mathematical expression.

$$S = {}^{2C_I}\!/_{C_x + C_y} \qquad (1)$$

where S is the similar content. C_I is the number of common words found in both files. C_x is the total count of words in file x. C_y is the total count of words in file y. for example:

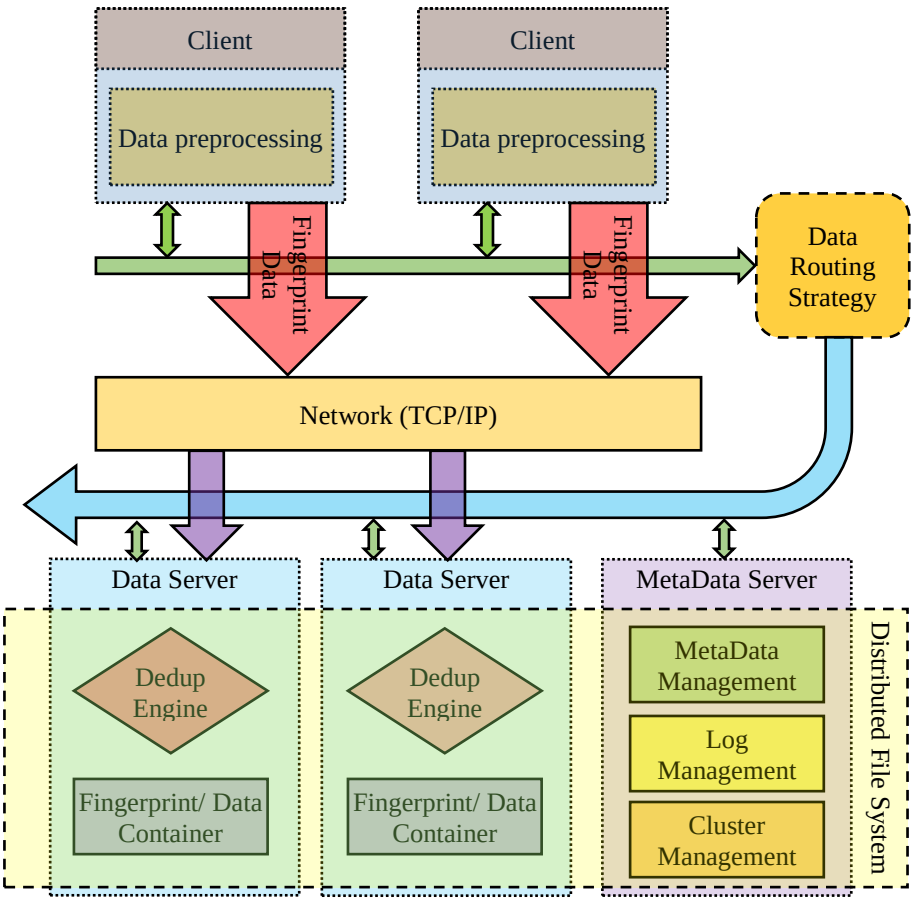

Figure 3: The working architecture of GA based proposed algorithm

Ram is a boy.
He is a boy.

Each set has four words, the intersection of these two sets results three elements i.e is, a, boy. The number of elements in each string is 4. Then Sϕrenson-Dice coefficient based similarity is:

Similarity S = $^{2*3}/_{4+4} = 0.75\ i.e.\ 75\%$

The same method also works for the finding similarity between words. The various available models like unigram/ bigram/ trigram or complete string matching are used for searching common elements between two streams.

n i g h t
n a c h t

The set of bigrams is searched in each word

ni ig gh ht
na ac ch ht

Each set has four elements, the intersection of these two sets has only one element 'ht'. The similarity is calculated based on Sϕrenson-Dice coefficient

Similarity S = $^{2*1}/_{4+4} = 0.25\ i.e.\ 25\%$

The stopping criteria for GA algorithm are the optimal value of dynamic single divisor 'D'. Good chunk-size variances improve redundancy detection by minimum computational overhead. Therefore, the value of divisor 'D' should exist in the pre-defined range where it can provide good size chunks. Sϕrenson-Dice coefficient is used to decide the range i.e. the minimum and maximum value for divisor according to the similarity between datasets. Initially, the min_value and max_value is decided by the minimum and maximum value of the similarity among the chromosomes respectively. The worst fit and best fit chromosomes are selected. The similarity value of worst-fit fixes the lower-bound (i.e. min value) of the chunk after multiplying by a factor and resulted value is represented by min_value. Similarly, the upper bound of the chunk is also calculated by multiplying a factor into similarity value of best-fit chromosome. The value of factor/s depends on the nature of the problem. Minimum and maximum values are the range in which the value of dynamic divisor can exist. After that, a random value for divisor 'D' is taken and proposed GA algorithm is applied on it. After each iteration, the value of divisor 'D' is updated and checked until the stopping criteria are not met.
For example The similarity value of the genetic population is calculated and let the worst-fit chromosome's similarity value is 25% and best-fit chromosome's similarity value is 65%. Let factor1 = 2 and factor2 = 40 is taken then min_value = 25 * 2 = 50 and max_value 65 * 40 = 2600, therefore, the value of dynamic divisor D should lie between 50 to 2600 bytes. The main advantage of this algorithm is its flexibility means the range of divisor D i.e. min_value and max_value can be adjusted according to nature of the problem.

5.1 Algorithm

Begin

1. divide dataset into the genetic population/chromosomes
2. randomly initialize the position to each chromosome
3. while *maximumcriteria* or *stoppingcondition* is not met do
   - 3.1 evaluate the *fitness* of chromosomes according to eqn. 1
   - 3.2 apply *roulette-wheel* method to select chromosomes from the population
   - 3.3 perform *crossover* on selected chromosome to generate new population
   - 3.4 perform *mutation* on a few chromosomes of new generation
   - 3.5 replace worst fit chromosomes with new population and *update* divisor D.

repeat from step 3.

5.2 Flowchart

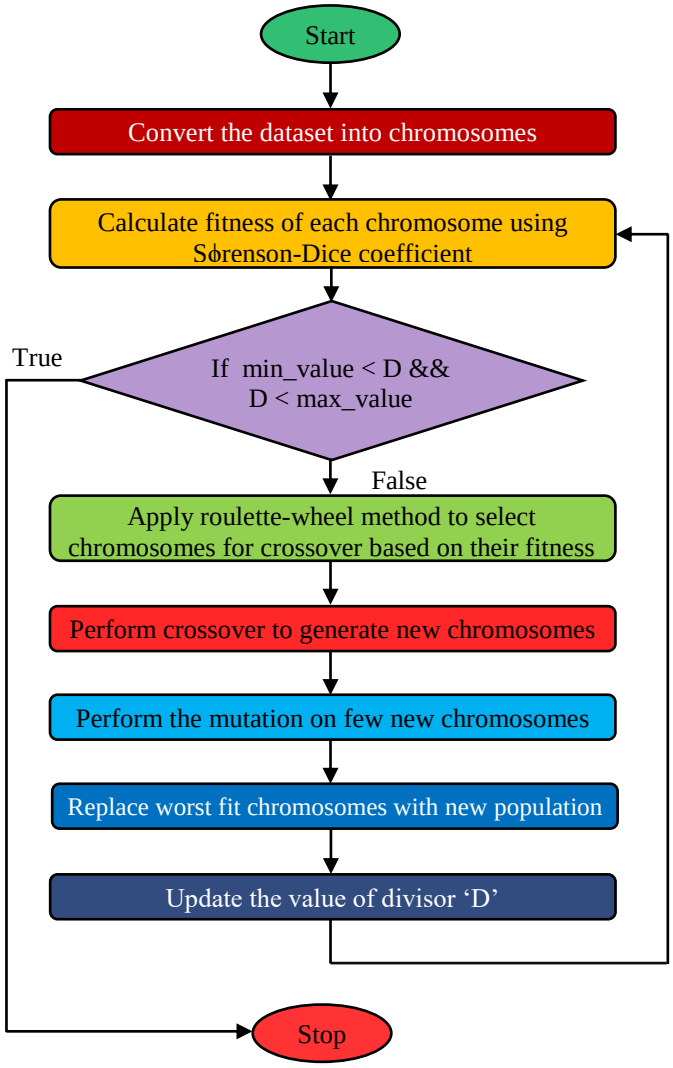

Figure 4: The flowchart of the proposed algorithms' chunking mechanism

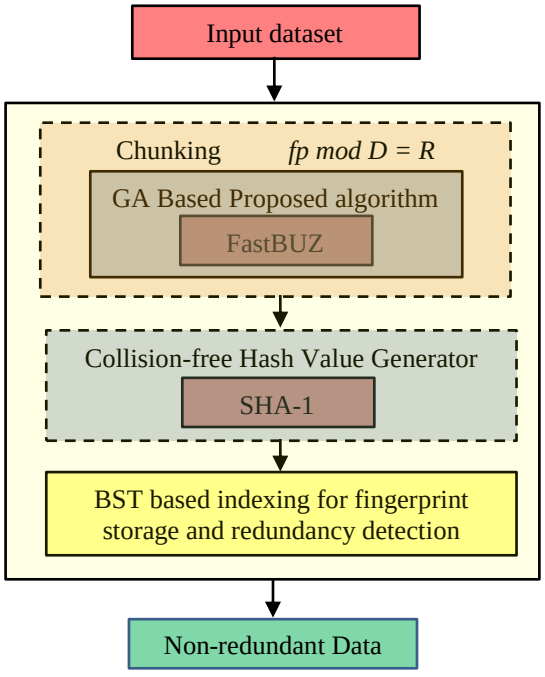

Figure 5: Step-by-step working of data deduplication system for proposed GA-based algorithm.

Indexing is an important phase where fingerprints are organized in a structured way. It is performed to check the uniqueness of data chunks and store or eliminate them. The techniques like DDFS [35] and Sparse Indexing [41] provides a powerful indexing mechanism but their number of comparisons are high. Binary search tree based indexing is gaining good attention in image vision task [40]. Binary search tree is a well-organized structure where lesser elements are always arranged in the left of root and greater elements are always fixed in the right of the root. It has searching and updating time of $\theta(\log n)$. Hence by applying this scheme, the deduplication process will lead to fast searching and updating. The same indexing is used in proposed system to reduce time complexity and number of comparisons.

MAP Function:

1. Map function reads the input data stream.
2. With the help of proposed GA based algorithm, the input data stream is split into dynamic variable size chunks. In chunking, the fastBUZ rolling hash function is used due to its less CPU overhead.
3. Store the chunks and generate the fingerprints for each chunk using SHA-1.

Reduce function:

1. Read the fingerprint of each chunk.
2. Compare the fingerprint by BST based index tree.
3. Store the unique chunks and update BST index tree by their fingerprints.
4. Eliminate the redundant chunk and reference them by old chunk.

6. Experimental Results and Analysis

The effectiveness and efficiency of the proposed system are evaluated by performing experiments on three different sized datasets. Three real-world datasets: VMDK, Linux, and Quanto are used for evaluation purpose. Each dataset is divided into variable-size chunks. All the experiments are conducted on a single node cluster of Hadoop system. BSW and TTTD are designed for primary and secondary storage (backup) but today data volume is very large and stored on distributed storage.

The specification used for experiments are given below:

- Operating System: Linux
- Version: Ubuntu 17.04LS
- Tool: Hadoop 2.8.0
- CPU: Intel® Core i3 (3.2GHz)
- RAM: 6GB
- HDD: 2TB, Seagate, 7200RPM

VMDK, a standard dataset, is commonly used in real-world [16]. It consists of 102 full backups. Each backup is 14.48GB on average and 89-95% are identical to its adjacent backups. Each backup contains nearly 16% self-referenced chunks and thus out-of-order containers are dominant. Linux [17] is a generally used public dataset. 258 consecutive versions of unpacked Linux kernel sources are zipped together. The average size of each version is 413MB. The consecutive versions are generally 99% identical. Quanto is a small-sized dataset having size 464MB only. 16 consecutive back-ups of general files are tar together. The average size of each version is 25MB. Its consecutive versions are generally 90% identical. All the experiments on each dataset are performed independently.

The $1/10^{th}$ of the dataset is taken to form the chromosomes and number of chromosomes and their size, both are decided according to size of the dataset. In short, the number of chromosomes are proportional to the size of the dataset. The selection criterion is decided to get maximum deduplication ratio at the cost of minimum metadata overhead. Initially, deduplication ratio and metadata overhead are calculated. The crossover operation is performed on selected chromosomes that are selected by roulette-wheel method [39]. The selection rate is moderate i.e. 10-15% of total population. The two-point crossover is applied where first and last four bits remain same and rest of the portion is swapped with the second one. After that mutation is performed on the chromosomes where two random bits are swapped together. Mutation rate should be less i.e. up to 5% only. Only the limited number of chromosomes are mutated i.e. one-tenth of the new population. After this, the same processes i.e. fitness calculation, chromosomes selection, crossover, and mutation are performed repeated on number of times. It is directly related to how much better solution is required. Higher the number of iterations, better the deduplication ratio. There should be a good trade-off between number of iteration and deduplication ratio because unnecessary iteration leads to computational overhead. As the optimal value of 'D' is retrieved, the iteration process should be stopped and this value is used to split the dataset into the chunks. The cryptographically secured hash signature (i.e. SHA-1) is applied to generate the fingerprints from chunks. Then these fingerprints are indexed in the form of binary search tree. By this way, complete deduplication process is performed on the datasets.

The characteristics of these datasets are shown in table 1. Table 2 shows the parameter configuration for the BSW [8], TTTD [9], and the proposed algorithm. Chunk size distributions of BSW, TTTD and the proposed algorithm is shown in table 3. The deduplication ratio of BSW, TTTD, and the proposed algorithm is shown in table 4. In table 5, the number of disk I/O operations for the proposed algorithm are compared with DDFS [35]. Figure 6 shows the graph for chunk size distribution of BSW, TTTD and the proposed algorithm. The graph for deduplication ratio analysis of BSW, TTTD, and proposed algorithm is shown in figure 7. Figure 8 shows the graph of percentage of total number of comparisons in DDFS and proposed algorithm.

Table 1: Characteristics of VMDK, Linux, and Quanto dataset for proposed algorithm

Dataset Name	Total Size	Number of back-ups	Deduplication ratio	Average chunk size
VMDK	0.72TB	51	53.37	10.3KB
Linux	104GB	258	78.13	11.9KB
Quanto	464MB	16	79.07	11.5KB

Table 2: Parameters configuration for the BSW, TTTD and proposed algorithm

Algorithm	Window Size (Bytes)	Main Divisor	Second Divisor	Maximum Threshold	Minimum Threshold
BSW	48	1000	N/A	N/A	N/A
TTTD	48	540	270	2800	460
Proposed	48	470	N/A	2300	600

Table 3: Chunk size distributions of the BSW, TTTD, and the proposed algorithm

Interval (Bytes)	BSW			TTTD			Proposed		
	VMDK	*Linux*	*Quanto*	*VMDK*	*Linux*	*Quanto*	*VMDK*	*Linux*	*Quanto*
<48(%)	0	0	NA	01.72	01.03	2.86	0	0	NA
48-459(%)	35.29	40.82	42.63	40.16	31.76	43.32	3.15	2.63	3.67
460-799(%)	18.72	20.01	21.45	27.23	21.68	28.57	10.34	9.81	11.43
800-1199(%)	15.46	14.59	15.01	13.82	14.55	13.59	30.66	30.11	31.37
1200-1599(%)	11.13	8.17	10.37	7.58	9.89	6.28	25.47	26.12	25.74
1600-1999(%)	7.85	4.56	6.18	4.03	7.36	2.47	18.22	20.07	18.01
2000-2399(%)	4.21	3.79	3.99	2.91	6.73	1.41	8.73	7.18	6.85
2400-2799(%)	2.97	2.87	0.29	2.43	5.77	1.19	2.97	2.64	2.17
>=2800(%)	4.37	5.19	0.08	0.12	1.23	0.31	0.46	1.44	0.76

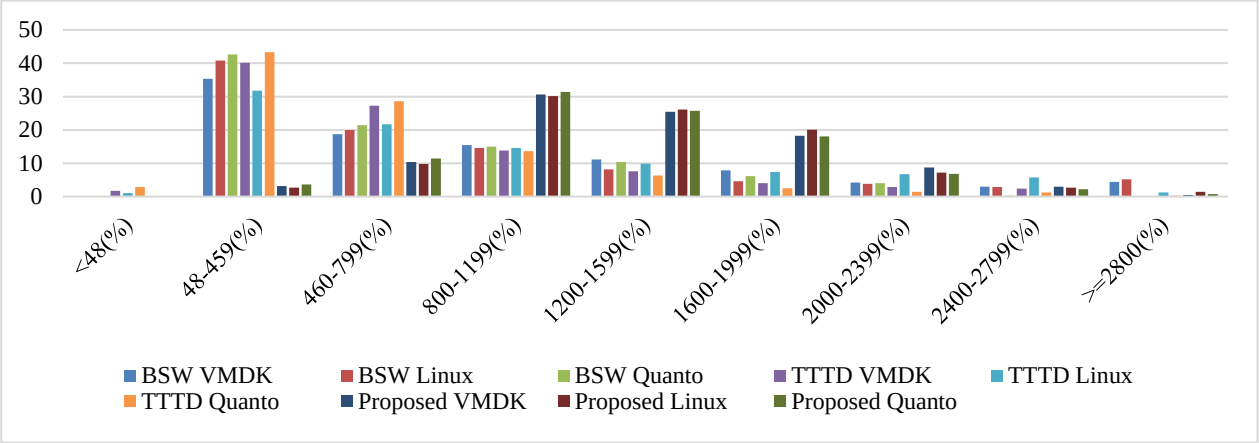

Figure 6: Chunk size distribution of BSW, TTTD, and the proposed algorithm

Table 4: Deduplication ratio of the BSW, TTTD, and the proposed algorithm

Interval (Bytes)	BSW			TTTD			Proposed		
	VMDK	*Linux*	*Quanto*	*VMDK*	*Linux*	*Quanto*	*VMDK*	*Linux*	*Quanto*
<48(%)	NA	NA	NA	0.29	0.35	0.32	0	0	NA
48-459(%)	0.63	0.71	0.67	0.70	0.76	0.73	0.80	0.82	0.81
460-799(%)	0.58	0.64	0.61	0.62	0.70	0.66	0.77	0.81	0.79
800-1199(%)	0.52	0.60	0.56	0.58	0.66	0.62	0.74	0.78	0.76
1200-1599(%)	0.45	0.54	0.49	0.51	0.57	0.54	0.69	0.75	0.72
1600-1999(%)	0.38	0.47	0.42	0.44	0.50	0.47	0.65	0.71	0.68
2000-2399(%)	0.32	0.41	0.35	0.38	0.44	0.41	0.60	0.66	0.63
2400-2799(%)	0.26	0.34	0.29	0.37	0.41	0.39	0.51	0.57	0.55
>=2800(%)	0.19	0.26	0.21	0.34	0.38	0.36	0.39	0.46	0.43

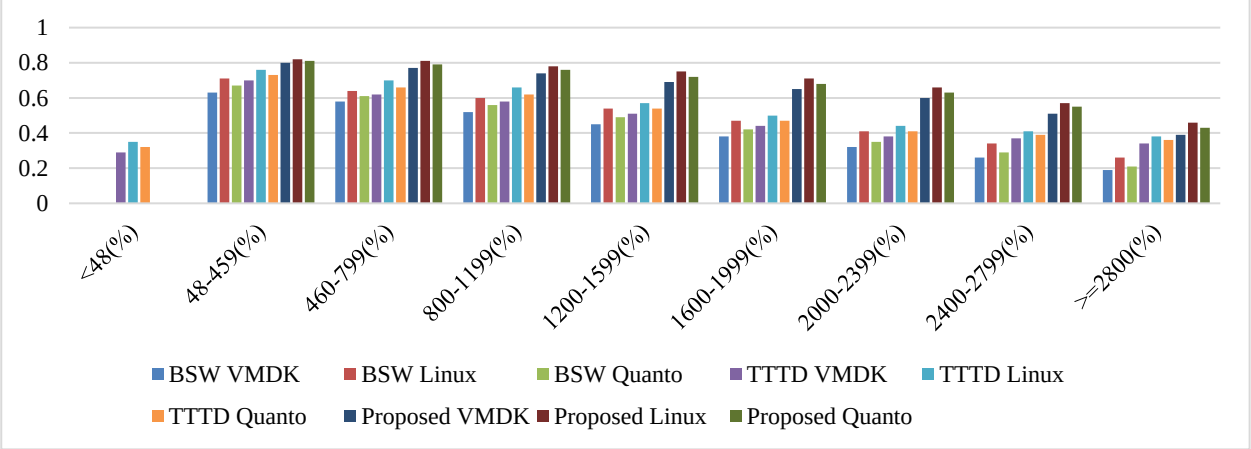

Figure 7: Deduplication ratio analysis of the BSW, TTTD and the proposed algorithm

Table 5: Number of disk I/O operations performed with or without summary vector and locality preserved caching, and proposed BST based indexing.

	VMDK		Linux		Quanto	
	# disk I/Os	*% of total*	*# disk I/O*	*% of total*	*% disk I/O*	*% of total*
No Summary vector and No Locality Preserved Caching	90,208,831	100%	12,092,322	100%	5,368	100%
Summary Vector Only	75,207,102	83.37%	9,858,870	81.53%	4,438	82.69%
Locality Preserved Caching	15,894,796	17.62%	2,280,611	18.86%	978	18.23%
Summary Vector and Locality Preserved Caching	893,067	0.99%	55,624	0.46%	30	0.57%
Proposed Algorithm (BST based Indexing)	6,440,910	7.14%	1,799,337	14.88%	449	8.37%

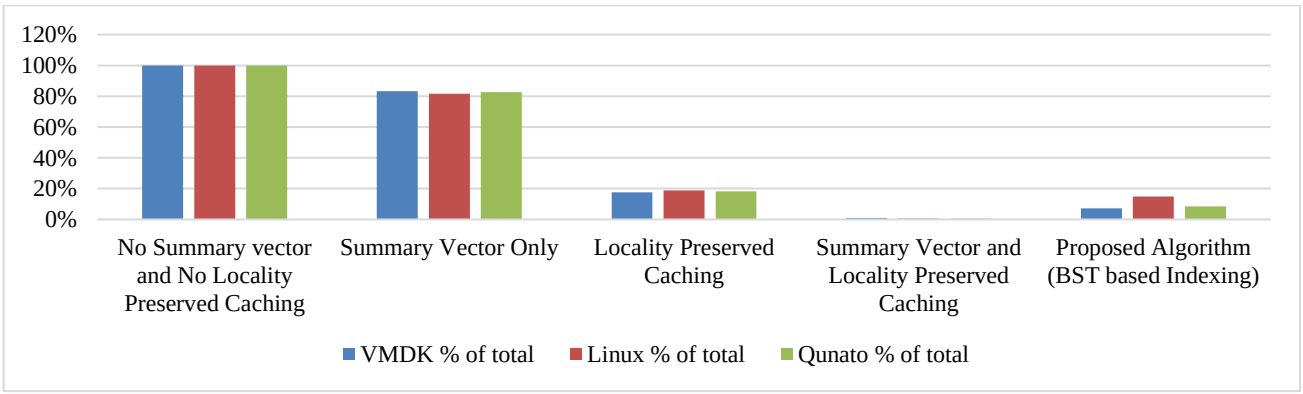

Figure 8: Percentage of total number of comparisons of DDFS and proposed algorithm.

7. Conclusion and Future Scope

In nutshell, the chunking schemes face the problem of very large and small chunk size. Another problem is about high disk I/O operations in indexing. The proposed approach mainly concentrates on chunking and indexing in distributed data deduplication system. The proposed approach uses GA to find the value of dynamic single divisor D for cut-points of the chunk. GA's main focus is to achieve good deduplication ratio but it takes a little bit extra time. The systems like DDFS, Sparse-indexing scheme work mainly on decreasing the number of comparisons and their average comparisons are 18% and 26% respectively. However, there is need to maintain a separate index for each node. The proposed algorithm takes only 7-15% comparisons; this is a remarkable achievement. If BST-indexing is applied on large scale system, it may not offer full performance due to very large index size. Moreover, DDFS and Sparse-indexing work on a single system and proposed model work on the distributed system. Finally, the result shows that the GA-based proposed system achieved better deduplication ratio and indexing than earlier systems. In future, other evolutionary techniques or advanced versions of GA may be applied to optimize the computational cost.

References

[1] D. Reinsel, J. Gantz, and J. Rydning, "Data Age 2025: The Evolution of Data to Life-Critical," Seagate, An IDC White Paper2017.

[2] Q. He, Z. Li, and X. Zhang, "Data deduplication techniques," in *Future Information Technology and Management Engineering (FITME), 2010 International Conference on*, 2010, pp. 430-433.

[3] C. Bo, Z. F. Li, and W. Can, "Research on chunking algorithms of data de-duplication," in *Proceedings of the 2012 International Conference on Communication, Electronics and Automation Engineering*, 2013, pp. 1019-1025.

[4] V. Henson, "An Analysis of Compare-by-hash," in *HotOS*, 2003, pp. 13-18.

[5] S. He, C. Zhang, and P. Hao, "Comparative study of features for fingerprint indexing," in *Image Processing (ICIP), 2009 16th IEEE International Conference on*, 2009, pp. 2749-2752.

[6] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, *et al.*, "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, vol. 104, pp. 1681-1710, 2016.

[7] F. Guo and P. Efstathopoulos, "Building a High-performance Deduplication System," in *USENIX annual technical conference*, 2011.

[8] A. Venish and K. Siva Sankar, "Study of Chunking Algorithm in Data Deduplication," in *International Conference on Soft Computing Systems (ICSCS)*, 2016, pp. 13-20.

[9] K. Eshghi and H. K. Tang, "A framework for analyzing and improving content-based chunking algorithms," *Hewlett-Packard Labs Technical Report TR*, vol. 30, 2005.

[10] T.-S. Moh and B. Chang, "A running time improvement for the two thresholds two divisors algorithm," in *Proceedings of the 48th Annual Southeast Regional Conference*, 2010, p. 69.

[11] I. Lkhagvasuren, J. M. So, J. G. Lee, C. Yoo, and Y. W. Ko, "Byte-index Chunking algorithm for data deduplication system," *International Journal of Security and its Applications*, vol. 7, pp. 415-424, 2013.

[12] W. Xia, Y. Zhou, H. Jiang, D. Feng, Y. Hua, Y. Hu, *et al.*, "FastCDC: a Fast and Efficient Content-Defined Chunking Approach for Data Deduplication," in *USENIX Annual Technical Conference*, 2016, pp. 101-114.

[13] J. J. Grefenstette, "How genetic algorithms work: A critical look at implicit parallelism," in *Proc. 3rd Int. Joint Conf. on Genetic Algorithms (ICGA89)*, 1989.

[14] S. Quinlan and S. Dorward, "Venti: A New Approach to Archival Storage," in *FAST*, 2002, pp. 89-101.

[15] T. White, *Hadoop: The definitive Guide*: O'Reilly Media, Inc, 2012.

[16] V. Tarasov, A. Mudrankit, W. Buik, P. Shilane, G. Kuenning, and E. Zadok, "Generating Realistic Datasets for Deduplication Analysis," in *USENIX Annual Technical Conference*, 2012, pp. 261-272.

[17] T. L. Foundation, "The Linux Kernel Archives," ed, 2017.

[18] M. Oberhumer, "LZO real-time data compression library," in *User Manual for LZO*, 0.28 ed, 1997.

[19] M. R. Nelson, "LZW data compression," *Dr. Dobb's Journal*, vol. 14, pp. 29-36, 1989.

[20] A. Z. Broder, "Some applications of Rabin's fingerprinting method," in *Sequences II*, ed: Springer, 1993, pp. 143-152.

[21] J. D. Cohen, "Recursive hashing functions for n-grams," *ACM Transactions on Information Systems (TOIS)*, vol. 15, pp. 291-320, 1997.

[22] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur, "Single instance storage in Windows 2000," in *Proceedings of the 4th USENIX Windows Systems Symposium*, 2000, pp. 13-24.

[23] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," in *ACM SIGOPS Operating Systems Review*, 2001, pp. 174-187.

[24] C. Yu, C. Zhang, Y. Mao, and F. Li, "Leap-based content defined chunking—theory and implementation," in *Mass Storage Systems and Technologies (MSST), 2015 31st Symposium on*, 2015, pp. 1-12.

[25] E. Kruus, C. Ungureanu, and C. Dubnicki, "Bimodal Content Defined Chunking for Backup Streams," in *Fast*, 2010, pp. 239-252.

[26] I. Lkhagvasuren, J. M. So, J. G. Lee, and Y. W. Ko, "Multi-level Byte Index Chunking Approach for File Synchronization," 2013.

[27] Y. Zhang, H. Jiang, D. Feng, W. Xia, M. Fu, F. Huang, *et al.*, "AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*, 2015, pp. 1337-1345.

[28] D. Teodosiu, N. Bjorner, Y. Gurevich, M. Manasse, and J. Porkka, "Optimizing file replication over limited-bandwidth networks using remote differential compression," 2006.

[29] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," *ACM Transactions on Storage (TOS)*, vol. 7, p. 14, 2012.

[30] J. Black, "Compare-by-Hash: A Reasoned Analysis," in *USENIX Annual Technical Conference, General Track*, 2006, pp. 85-90.

[31] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full SHA-1," in *Annual International Cryptology Conference*, 2017, pp. 570-596.

[32] A. W. Appel, "Verification of a cryptographic primitive: SHA-256," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 37, p. 7, 2015.

[33] K. Srinivasan, T. Bisson, G. R. Goodson, and K. Voruganti, "iDedup: latency-aware, inline data deduplication for primary storage," in *FAST*, 2012, pp. 1-14.

[34] J. Wei, H. Jiang, K. Zhou, and D. Feng, "MAD2: A scalable high-throughput exact deduplication approach for network backup services," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, 2010, pp. 1-14.

[35] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the Disk Bottleneck in the Data Domain Deduplication File System," in *Fast*, 2008, pp. 1-14.

[36] ZFS, ed, 2012.

[37] I. Drago, M. Mellia, M. M Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside dropbox: understanding personal cloud storage services," in *Proceedings of the 2012 Internet Measurement Conference*, 2012, pp. 481-494.

[38] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*: MIT press, 1992.

[39] N. M. Razali and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving TSP," in *Proceedings of the world congress on engineering*, 2011, pp. 1134-1139.

[40] L. Brown and L. Gruenwald, "Tree-based indexes for image data," *Journal of Visual Communication and Image Representation*, vol. 9, pp. 300-313, 1998.

[41] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezis, and P. Camble, "Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality," in *Fast*, 2009, pp. 111-123.